FairWind++

FairWind++ is an embedded operating environment built for ships, boats and other seagoing vehicles. It extends and improves the idea of on-board systems, introducing endless possibilities: spacing from the sea and navigation data management to the execution of complex connected general purpose applications. With this perspective, FairWind++ provides an easy and efficient navigation experience to the sailor, much closer to what one is used to seeing on smartphones and other modern portable devices. FairWind++ is based on modern, open technologies and provides a full software development kit for developing apps and other pieces of software for the FairWind ecosystem. Safety, entertainment, navigation, administration, data analysis and much more, these are the capabilities of FairWind++.

1.0 Introduction

Vehicles have been a fundamental part of people's life for many decades now and have been evolving too during these years. Cars, for instance, have been through a transformative process for many years now, and are still evolving. Driver assisting technologies, self-driving prototypes, entertainment systems have invaded the market, making a car not just a simple transport tool, not anymore, but a complex and intelligent ecosystem. Cars can now drive themselves, play media contents, show directions, communicate with other cars and people, collect and analyze useful information about fuel consumption, driver's and passenger's state and so on.

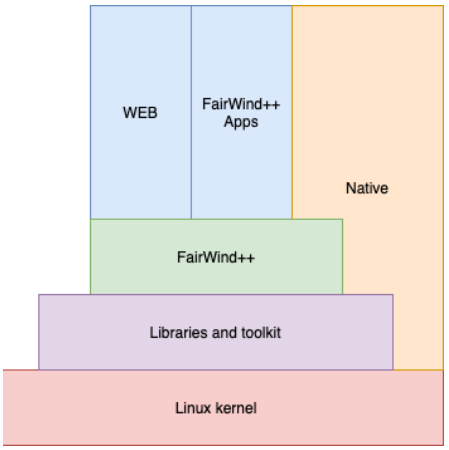
Cars are not the only tools involved in this evolution process. Seagoing vehicles for instance have been evolving too. These vehicles need advanced and precise instrumentation to guarantee its passengers' safety. Weather monitoring, navigation direction and speed, water pressure… This data, almost certainly provided by sensors and other specific instrumentation, needs to be managed properly by on-board systems to make the sailor's work possible. That's where FairWind++ comes in: it can store and manage this data with clarity and precision, while also providing features people are already accustomed to when using their smartphones and/or personal computers. FairWind++ is packed with many applications and utilities for both sailors and passengers, making their trip easier, safer and much more enjoyable.

FairWind++ works best if connected to the internet, having the possibility to gather information from various dedicated servers. However, an internet connection is not required to use the FairWind++ device.

FairWind++'s capabilities can be expanded with external peripherals and sensors, which can be accessed and used on the main device through their own specific software and applications.

1.1 Internal Structure

FairWind++ has a simple and straightforward structure. As said before, FairWind++ is not an operating system, but rather an operating environment. That means that it is nearer to the concept of desktop environment: it runs in full screen on top of an actual operating system, GNU/Linux in this case, and allows the user to launch other applications. FairWind++ is also capable of launching web applications and native Linux applications, extending the usage scenarios quite much. Interaction with the underlying hardware is done through two layers: first is the libraries layer and then the kernel. The hardware is accessed through external software that is not directly involved with the FairWind++ user experience.



FairWind++'s internal structure

1.2 Requirements

This section analyzes the FairWind++'s build process and requirements, useful to cover testing and developing scenarios.

FairWind++ is not an operating system by itself, since it runs on top of an existing GNU/Linux distribution. The underlying distribution doesn't matter much, as long as it provides a relatively updated Linux kernel and Qt5 bindings and dependencies. FairWind++ needs the full Qt5 SDK in order to work, with a focus on qt5base and qt5webengine packages, which provide the main libraries to build FairWind++'s UI and make it possible to run normal applications and web-based ones.

FairWind++ do not apply further limitations to the hardware to be used: as long as the hardware is supported by the Linux kernel, and by the Qt5 framework for that matter, it's also supported by FairWind++. Of course, FairWind++ is specifically designed to work on a wide landscape touch screen and other external hardware (peripherals, sensors…) would be necessary to take advantage of some of the main features of FairWind++.

FairWind++ build system was tested against Qt6 (6.2.0, specifically) but, due to breaking changes in some APIs from Qt5, some dependencies could not be built in the process. For this reason, it's recommended to build using Qt5 SDK and nothing more.

CMAKE, Git and the GCC compiler are other strictly required dependencies to build FairWind++ and should be installed before continuing. Gstreamer library is required to build and use FairWind++'s media components.

1.3 Build process

Building FairWind++ is possible, and has been done, on a variety of hardware and systems. Below there are instructions on how to gather the required dependencies and build FairWind++ successfully. These below are shell commands, so it is also required a basic understanding of the system's terminal emulator.

1.3.1 Ubuntu Linux

First of all, download and install all the required dependencies using the apt-get package system:

- sudo apt-get install build-essential cmake git qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools qtwebengine5-dev libqt5websockets5-dev qtmultimedia5-dev libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev libgstreamer-plugins-bad1.0-dev gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly gstreamer1.0-libav gstreamer1.0-doc gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-alsa gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio

Now, clone the FairWind++ project from its GitHub repository:

- git clone https://github.com/OpenFairWind/fairwindplusplus.git

Navigate to the cloned directory and prepare build environment:

- cd fairwindplusplus
- mkdir build
- cd build
- cmake -DCMAKE_PREFIX_PATH="/lib/qt5/" ..

Proceed with the build process:

- make

Run FairWind++:

- ./FairWind

1.3.2 Raspberry Pi OS

- Build QT 5.15.2 using the guide
  here: https://www.tal.org/tutorials/building-qt-515-raspberry-pi

- wget https://download.qt.io/official_releases/qt/5.15/5.15.2/single/qt-everywhere-src-5.15.2.tar.xz

- tar xf qt-everywhere-src-5.15.2.tar.xz

- git clone https://github.com/oniongarlic/qt-raspberrypi-configuration.git

- cd qt-raspberrypi-configuration && make install DESTDIR=../qt-everywhere-src-5.15.2

- sudo apt update

- sudo apt install build-essential libfontconfig1-dev libdbus-1-dev libfreetype6-dev libicu-dev libinput-dev libxkbcommon-dev libsqlite3-dev libssl-dev libpng-dev libjpeg-dev libglib2.0-dev libraspberrypi-dev

- sudo apt install bluez libbluetooth-dev

- sudo apt install libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-ugly gstreamer1.0-plugins-bad libgstreamer-plugins-bad1.0-dev gstreamer1.0-pulseaudio gstreamer1.0-tools gstreamer1.0-alsa

- sudo apt install libasound2-dev

- sudo apt install pulseaudio libpulse-dev

- sudo apt install libx11-dev libxcb1-dev libxext-dev libxi-dev libxcomposite-dev libxcursor-dev libxtst-dev libxrandr-dev libfontconfig1-dev libfreetype6-dev libx11-xcb-dev libxext-dev libxfixes-dev libxi-dev libxrender-dev libxcb1-dev libxcb-glx0-dev libxcb-keysyms1-dev libxcb-image0-dev libxcb-shm0-dev libxcb-icccm4-dev libxcb-sync-dev libxcb-xfixes0-dev libxcb-shape0-dev libxcb-randr0-dev libxcb-render-util0-dev libxcb-util0-dev libxcb-xinerama0-dev libxcb-xkb-dev libxkbcommon-dev libxkbcommon-x11-dev

- sudo apt install flex bison gperf libre2-dev libnss3-dev libdrm-dev libxml2-dev libxslt1-dev libminizip-dev libjsoncpp-dev liblcms2-dev libevent-dev libprotobuf-dev protobuf-compiler

- sudo apt install libgles2-mesa-dev libgbm-dev

- mkdir build cd build

- PKG_CONFIG_LIBDIR=/usr/lib/arm-linux-gnueabihf/pkgconfig:/usr/share/pkgconfig
- ../qt-everywhere-src-5.15.2/configure -platform linux-rpi4-v3d-g++ -v -opengl es2 -eglfs -no-gtk -opensource -confirm-license -release -reduce-exports -force-pkg-config -nomake examples -no-compile-examples -skip qtwayland -skip qtwebengine -no-feature-geoservices_mapboxgl -qt-pcre -no-pch -ssl -evdev-system-freetype -fontconfig -glib -prefix /opt/Qt/5.15.2 -qpa eglfs
- make
- make install
- Build qtwebengine using this guide: https://www.tal.org/tutorials/building-qtwebengine

1.3.3 MacOs

It is possible to build FairWind++ only on Intel powered Mac systems. On Apple Silicon powered Macs, the build process won't succeed, cause Qt5 (5.15.2, currently) still does not support the new processors. Qt6 (6.2.0) introduced initial support but as specified earlier, some FairWind++ components can't be built against Qt6.

First, let us get the required dependencies:

- brew install git cmake qt@5

Now, clone the FairWind++'s GitHub repository:

- git clone https://github.com/OpenFairWind/fairwindplusplus.git

Access the cloned directory and prepare the build environment:

- cd fairwindplusplus
- mkdir build
- cd build
- cmake -DCMAKE_PREFIX_PATH="/usr/local/opt/qt5/"

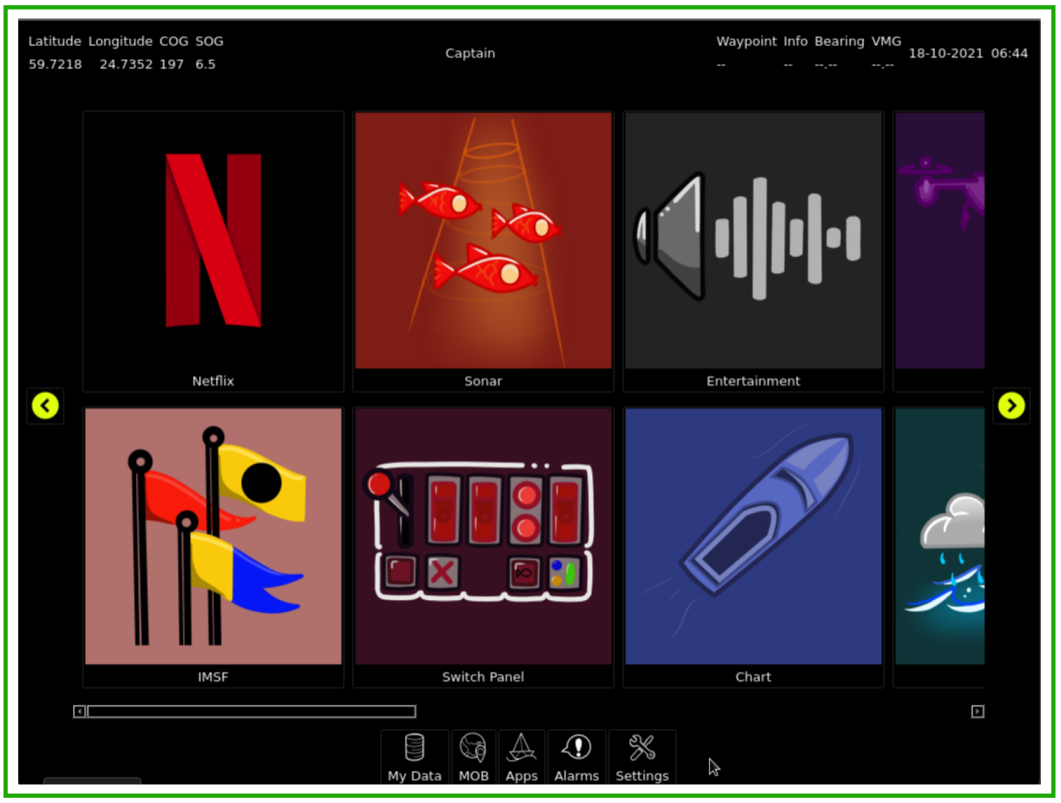
Proceed with the build process:

- make

Run FairWind++:

- ./FairWind

2.0 User Interface

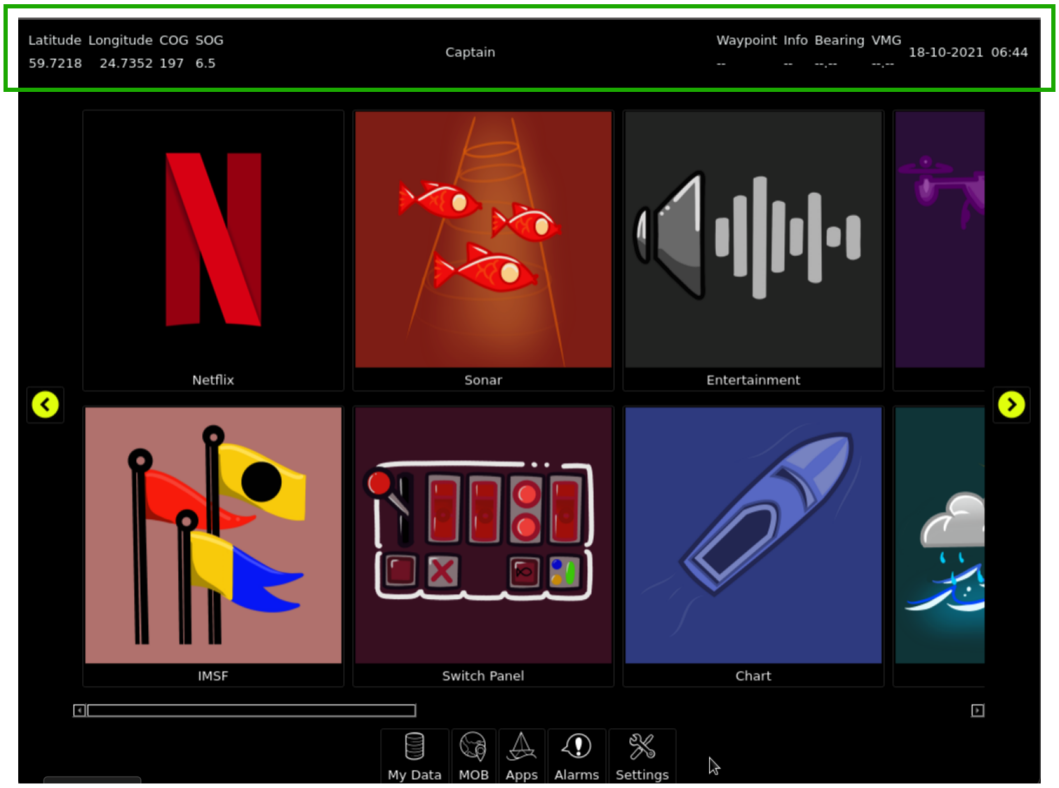


FairWind++'s main UI

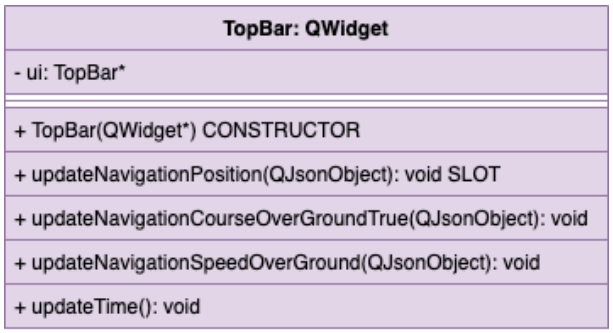
FairWind++ is powered by Qt5, a modern and powerful UI development framework. Qt5 allows developers to design and code complex applications only once using their language of choice (C, C++, QML, JavaScript…), and compile and run them on multiple systems without any changes, as long as the system in use provides the necessary Qt bindings and dependencies. FairWind++'s user interface is optimized for wide landscape touch displays. Its structure has been studied to provide crucial navigation information at a glance, dividing the available display space in different areas, each one with its purpose.

2.1 TopBar

As the name suggests, this is an area on top of the screen. It presents useful information, like navigation speed, current date and time, geographic coordinates and so on. The correctness of these information is ensured updating them constantly. FairWind++'s top bar emulates the features of other status bars already seen in many different flavors on the most common operating systems: Android, iOS, Windows.



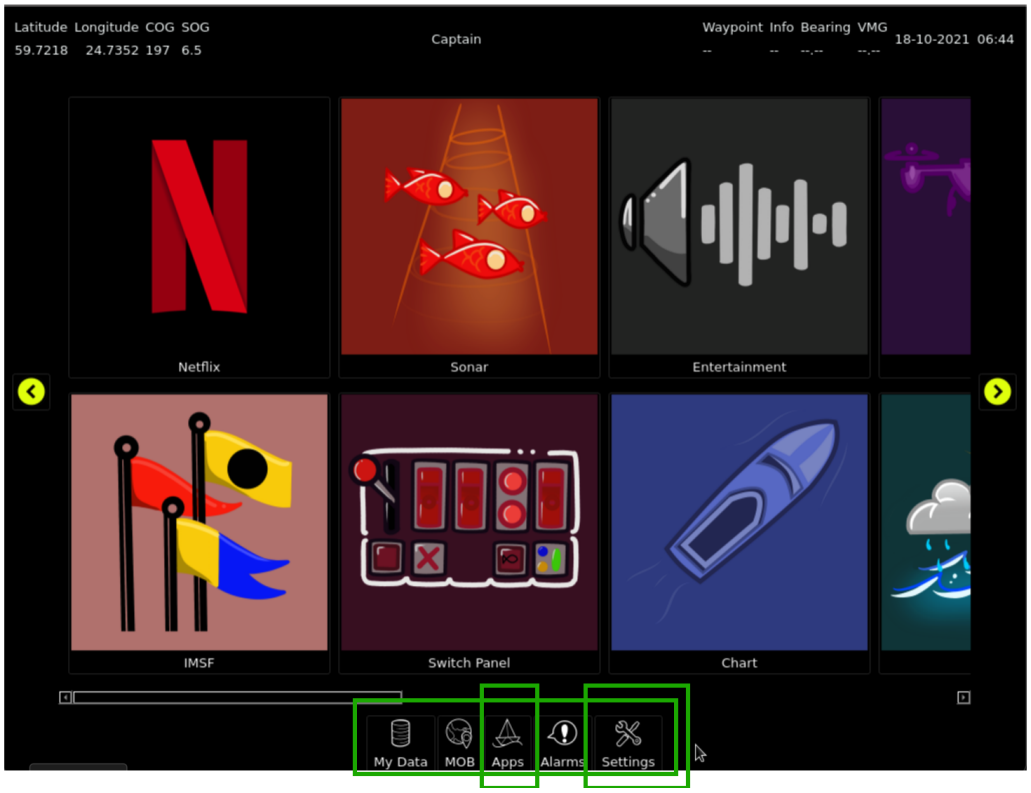
FairWind++'s top status bar



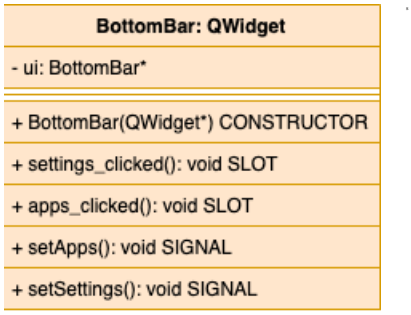
TopBar's class diagram

2.2 BottomBar

Again, as the name suggests, this area is located on the bottom of the screen. It doesn't directly provide any kind of information: its purpose is to let the sailor navigate through the different areas of the user interface with just quick and simple touches. The bottom bar can also be hidden when a high priority application is launched, in order to take advantage of all the available screen space.



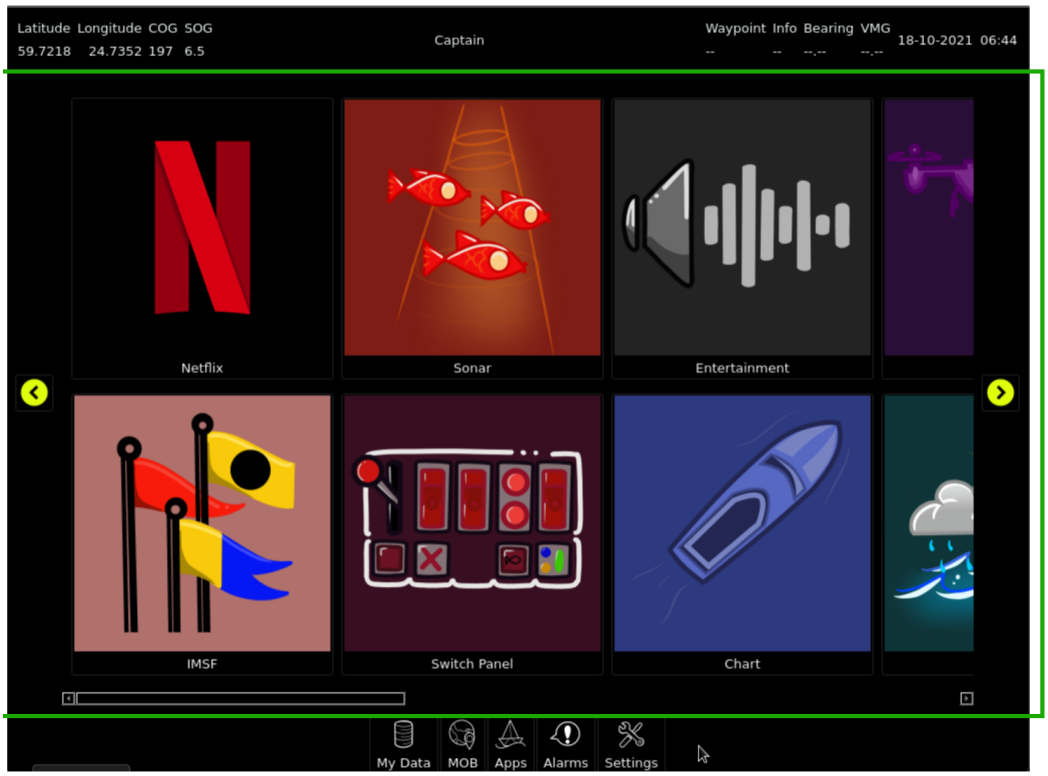
FairWind++'s bottom navigation bar



BottomBar's class diagram

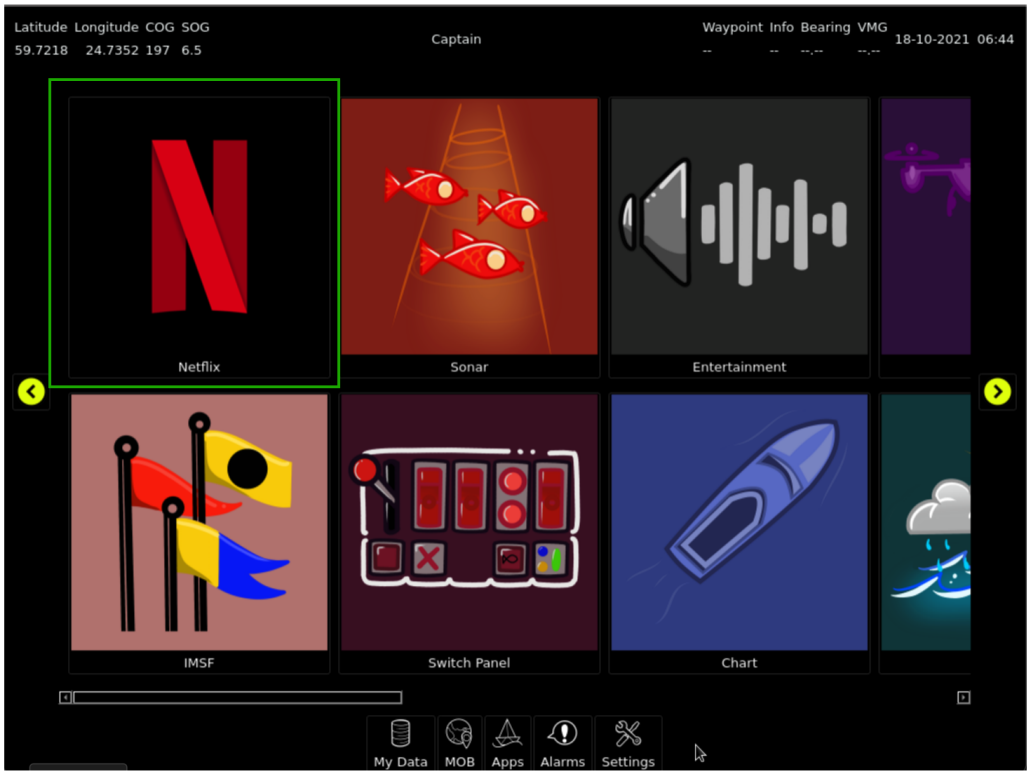
2.3 Applications

The center screen area can present different contents based on the user actions. The main content is a list containing all the installed applications: they're organized in a scrollable grid layout, allowing an easy navigation inside the list. When an application is launched, it will fill the entire center area, in order to show as much of the application's content as possible. Some applications can also be built as high priority, meaning that, once launched, they will take advantage of all the available space, even hiding the bottom bar.

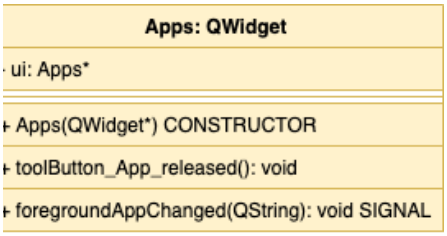


FairWind++'s center screen area

Every application will be displayed as a widget on the display, allowing the human user to interact. Once one of the widgets is clicked, the system will send a signal to the user interface; the user interface will then respond by changing the center screen area content with the application's content.



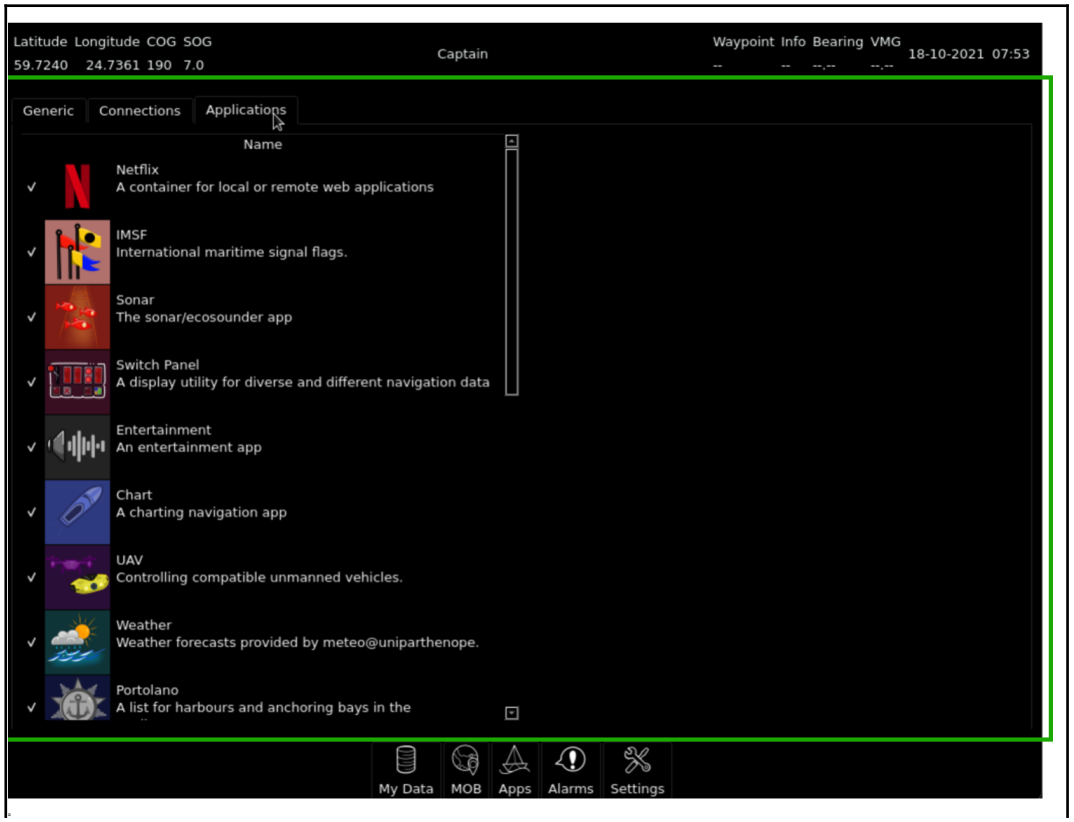
FairWind++'s apps view



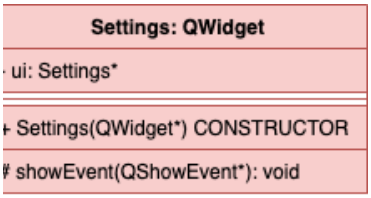
Apps' class diagram

2.4 Settings

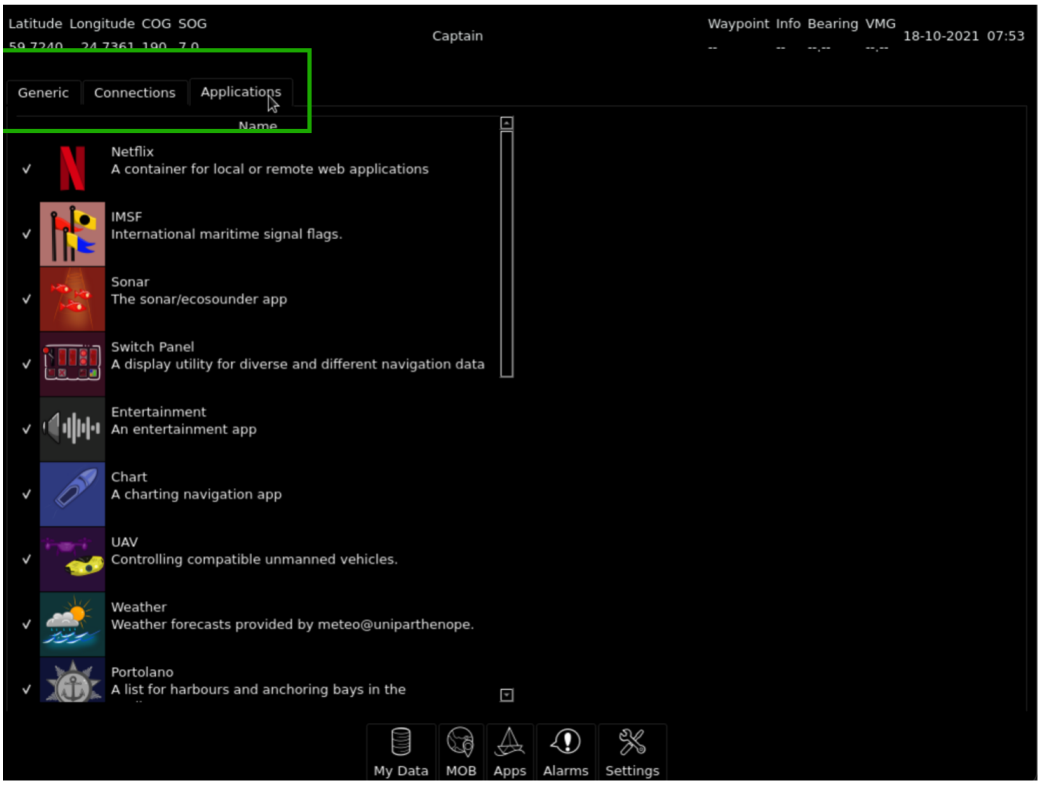
The center screen area can also present the FairWind++'s settings view. Here, the sailor can easily configure the device, navigating in three different tabs that categorize the available settings: generic, applications, and connections.



FairWind++'s settings view

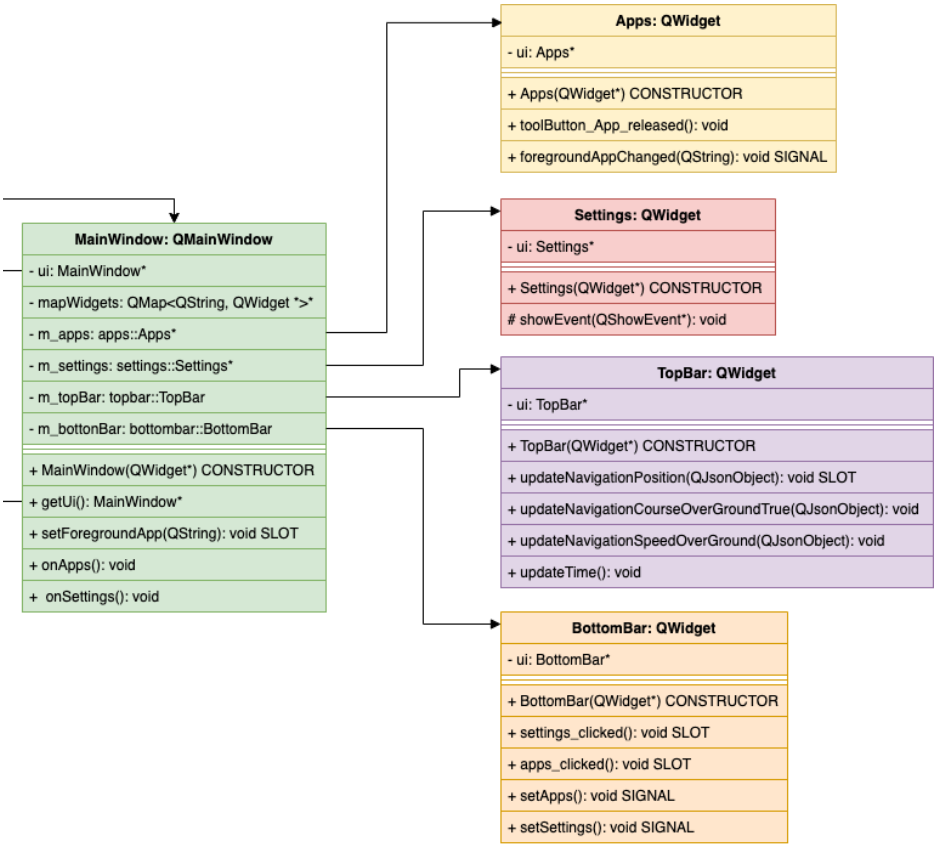


Settings' class diagram



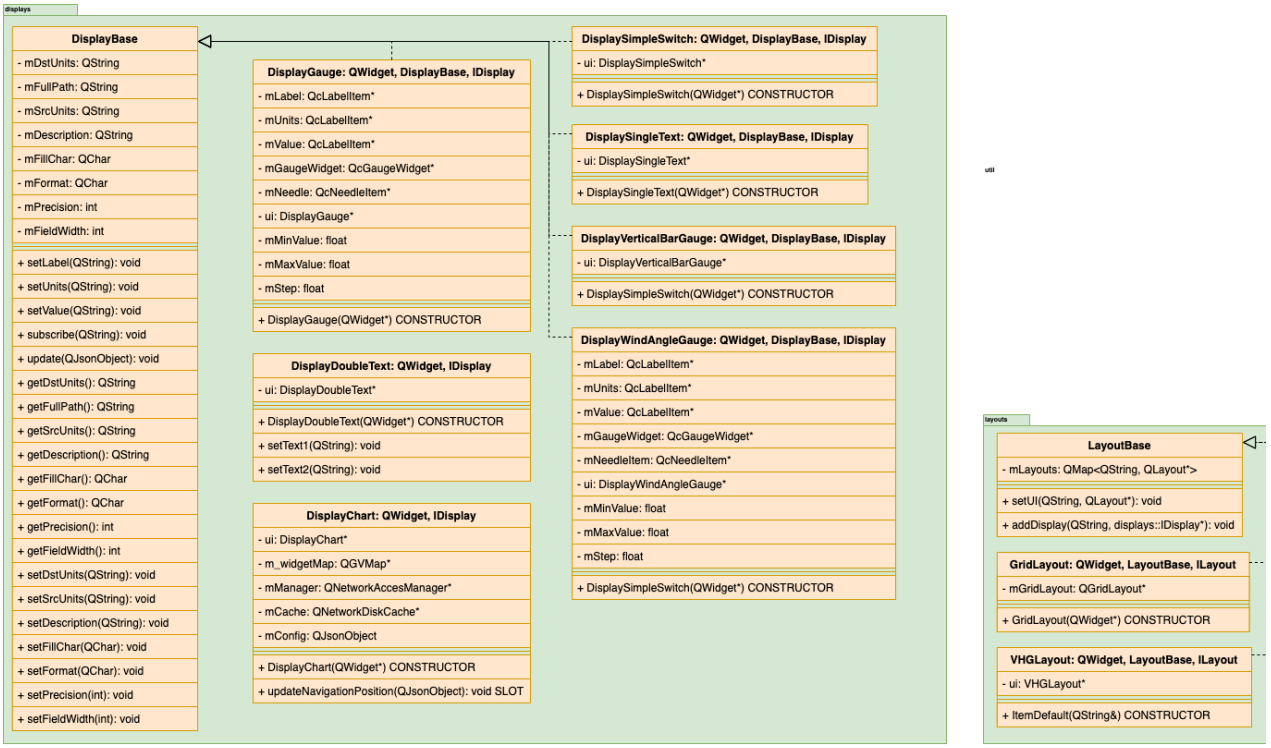
Settings tabs

The user interface is organized in blocks, which are presented inside a Qt main window application. Every block is loaded as a Qt widget and placed on the screen during the boot process.



MainWindow's class diagram

It’s also worth mentioning that FairWind++’s user interface is flexible and extendable: developers can create their own custom layouts and visual components in Qt5 and add them as they please to their applications.



Displays and Layout packages class diagrams

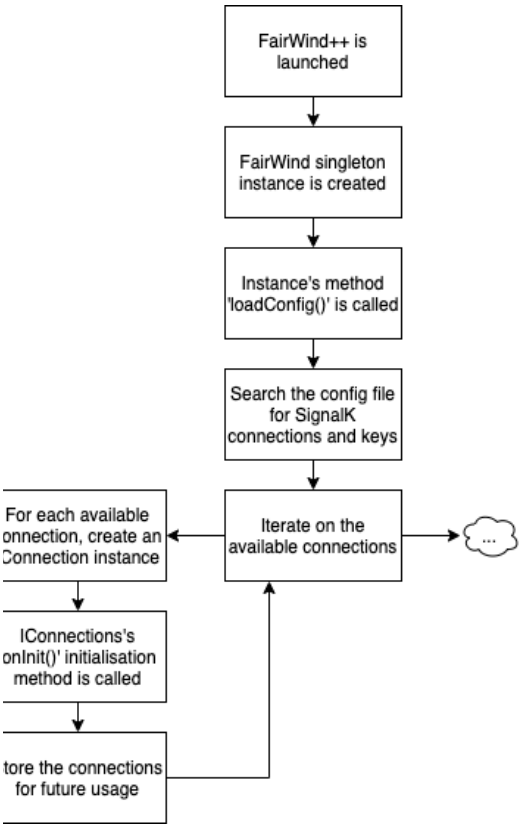
3.0 Connections

A connection in FairWind++ is a service that provides navigation data and a place to store the data gathered during FairWind++ everyday usage. Depending on the sailor's needs, a connection can provide a multitude of different data that can be stored, processed and displayed in the FairWind++ device: navigation speed, position, wind orientation, meteorological forecasts and so on.

Even though a connection is not required for FairWind++ to work as intended, it is highly recommended to use one anyway, in order to unlock the full potential of the software.

3.1 SignalK

The first service that FairWind++ can work with is SignalK. SignalK is a modern and open data format for marine use, capable of providing navigation information that can be shared, processed and displayed easily. A SignalK connection needs a SignalK key, to which FairWind++ will subscribe to get constantly updated data. All the keys shall be stored inside a configuration json file placed in an accessible position. The program will scan the file during the boot process looking for a 'connections' and a 'keys' keys, which both represent json objects containing all the information required by SignalK servers in order to process and satisfy FairWind++'s requests.



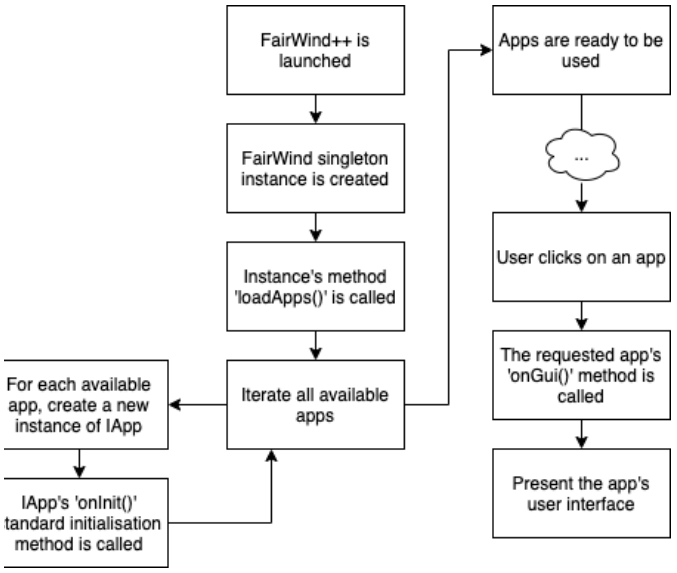
The configuration file shall provide the necessary keys, organizing them in a json object. A precise syntax is followed to get all the data correctly. There's an example of a working SignalK object for the configuration file below.

```
"SignalK":{
  "connections":[
    {
      "class":"fairwind::connections::SignalKAPIClient",
      "active":true,
      "url":"...",
      …
    },
    {
      "class":"fairwind::connections::SignalKWSClient",
      "active":true,
      "url":"..."
    }
  ],
  "keys":{
    "**navigation key**":{
     …
    }
  }
}
```

Each key FairWind++ subscribes to will provide determinate information.
In the absence of a valid connection, FairWind++'s data will be stored locally on the physical device; navigation information, however, won't be collected from the service.

4.0 Applications

Applications in FairWind++ are handled in an unorthodox but efficient way. During the boot process of the system, installed applications are automatically loaded, making them ready to be used by the user. Every application is always running in background, and the actual user interface is raised when the application is directly launched by the user, so avoiding long and painful load times that can even be dangerous in certain situations. Applications can also be configured and the configuration details are saved in a local file, placed in an accessible location, which is also loaded in turn during the boot process.



FairWind++'s apps loading process

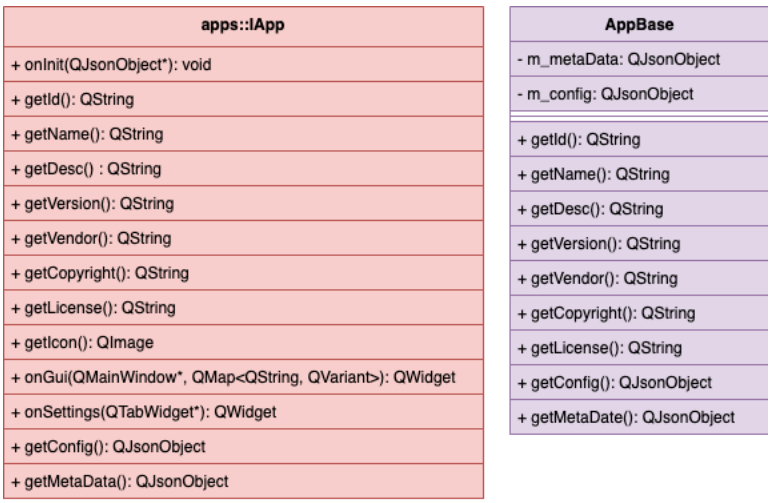
FairWind++'s applications are built and handled as Qt plugins and can cover all kinds of topics: data management, entertainment, on-board safety, seagoing driving assist and so on.
Some applications can also be built to have a high priority, meaning that they will fill the entire available screen space when launched, even hiding the bottom bar.

4.1 Building an application for FairWind++

FairWind++ provides a full software development kit, which makes it easy to design and develop applications aimed at the sailor's needs or at the passengers' experience. When building an application for FairWind++, the system in use has to provide the same dependencies required by the FairWind++'s build process itself.

When building a new FairWind++ application, it is important to remember how FairWind++ handles them at a code wise level. Every application running in FairWind++ is an instance of App (fairwind::App): this class is packed with many different private members used to store the application's details, in order to access them later rather quickly. These details can provide a variety of information about the application: name, version, license, description and so on. App class can be instantiated through two different constructors: one accepts an existing App instance as an input parameter and loads the application details directly from the provided instance, the second one accepts an object that implements the interface IApp (fairwind::app::IApp). IApp is an interface that defines the most basic methods every FairWind++ application should have: various getters for the application's details, an initialization method, a method called when the application's user interface needs to be raised and finally a method called when the application's settings need to be raised. Being defined by an interface, each one of these methods can be implemented according to the application's needs and peculiar aspects. Nevertheless, FairWind's SDK provides a default implementation for each one of IApp's methods. If the default implementations reflect the application's needs, then the developer can use them though the AppBase class. AppBase is a class that provides IApp's methods' default implementations.

In order to create a new FairWind++ application from scratch, it's mandatory to create a new class that identifies it. Basically, a FairWind++ application is nothing but a Qt5 plugin that needs to be loaded inside the FairWind++ ambient, therefore the application's class must inherit the QObject class (see Qt5 specific documentation). Now, to be properly handled by the FairWind++ ambient, it needs to provide the basic methods, so the application's class must inherit IApp too. Finally, if the default implementations are needed, the application's class should inherit AppBase too. At this point, the application has everything required to be handled properly by the FairWind++'s ambient.



IApp and AppBase class diagram

Once the application's class is finished and provides all the required components, FairWind++'s peculiar applications loading process comes in. During the loading process, FairWind++ will search for Qt5 generic plugins; if a plugin can be successfully casted to an IApp object, then it represents a FairWind++ application. The IApp instance is then passed to the App constructor as an input parameter. At the end of this process, every application will be loaded and will be represented as an App instance.

App's class diagram

FairWind++'s applications can cover almost any field: driving assistance, data gathering and management, on-board safety and even entertainment. An app can even be built specifically to work with some hardware components that can be installed on-board: sensors, cameras, communication devices and so on.